Section: **2.3**
Development Techniques

Description: This section describes some examples of development techniques that can be used with the Departmental software engineering methodology. The examples include high-level instructions on how to adapt the lifecycle stages to accommodate the development technique. *Exhibit 2.2-1, Adapting the Lifecycle,* shows how some development techniques can be used with the software lifecycle model. The examples provided here are not intended to be a comprehensive list of development techniques.

Segmented Development: Segmented development is most often applied to large software engineering projects where the project requirements can be divided into functional segments. Each segment becomes a separate project and provides a useful subset of the total capabilities of the full product. This segmentation serves two purposes: to break a large development effort into manageable pieces for easier project management and control; and to provide intermediate work products that form the building blocks for the complete product.

The lifecycle processes and activities are applied to each segment. The overall system and software objectives are defined, the system architecture is selected for the overall project, and a Project Plan for development of the first segment is written and approved by the system owner.

Segments are delivered to the system owner for evaluation or actual operation. The results of the evaluation or operation are then used to refine the content of the next segment. The next segment provides additional capabilities. This process is repeated until the entire software product has been developed. If significant problems are encountered with a segment, it may be necessary to reexamine and revise the project objectives, modify the system architecture, update the overall schedule, or change how the segments are divided.

Two major advantages of this approach are: the project manager can demonstrate concrete evidence that the final product will work as specified; and users will have access to, and use of, segments or functions prior to the delivery of the entire software product.

Spiral Development: Spiral development repeats the planning, requirements, and functional design stages in a succession of cycles in which the project's objectives are clarified, alternatives are defined, risks and constraints are identified, and a prototype is constructed. The prototype is evaluated and the next cycle is planned.

Spiral Development, continued:

The project objectives, alternatives, constraints, and risks are refined based on this evaluation; then, an improved prototype is constructed. This process of refinement and prototyping is repeated as many times as necessary to provide an incrementally firm foundation on which to proceed with the project.

The lifecycle activities for the Planning, Requirements Definition, and Functional Design Stages are repeated in each cycle. Once the design is firm, the lifecycle stages for System Design, Programming, and Integration and Testing are followed to produce the final software product.

Rapid Prototyping:

Rapid prototyping can be applied to any software development methodology (e.g., segmented, spiral). Rapid prototyping is recommended for software development that is based on a new technology or evolutionary requirements.

With the rapid prototyping technique, the most important and critical software requirements are defined based on current knowledge and experience. A quick design addressing those requirements is prepared, and a prototype is coded and tested. The purpose of the prototype is to gain preliminary information about the total requirements and confidence in the correctness of the design approach. Characteristics needed in the final software product, such as efficiency, maintainability, capacity, and adaptability might be ignored in the prototype.

The prototype is evaluated, preferably with extensive user participation, to refine the initial requirements and design. After confidence in the requirements and design approach is achieved, the final software is developed. The prototype might be discarded, or a portion of it used to develop the final product.

The normal software engineering documentation requirements are usually postponed with prototyping efforts. Typically, the project team, project stakeholders, and system owner agree that the prototype will be replaced with the actual software product and required support documentation after proof of the model. The software that replaces the prototype should be developed using the lifecycle processes and activities.

Iterative Technique:

The iterative technique is normally used to develop software products piece by piece. Once the system architecture and functional or conceptual design are defined and approved, system functionality can be divided into logically related pieces called "drivers."

Iterative Technique, continued:

In iterative fashion, the project team performs system design, code, unit test, and integration test activities for each driver, thereby delivering a working function of the product. These working functions or pieces of the software product are designed to fit together as they are developed. This technique allows functions to be delivered incrementally for testing so that they can work in parallel with the project team. It also enables other functional areas, such as documentation and training, to begin performing their activities earlier and in a more parallel effort. In addition, the iterative technique enables progress to be visible earlier, and problems to be contained to a smaller scope.

With each iterative step of the development effort, the project team performs the lifecycle processes and activities.

Rapid Application Development:

Rapid Application Development (RAD) is a method for developing systems incrementally and delivering working pieces every 3 to 4 months, rather than waiting until the entire project is programmed before implementation. Over the years, many information projects failed because by the time the implementation took place, the business had changed.

RAD employs a variety of automated design and development tools, including Computer-Aided Software Engineering (CASE), fourth-generation language (4GLs), visual programming, and graphical user interface (GUI) builders, which get prototypes up and running quickly. RAD focuses on personnel management and user involvement as much as on technology.

Joint Application Development:

Joint Application Development (JAD) is a RAD concept that involves cooperation between the designer of a computer system and the end user to develop a system that meets the user’s needs exactly. It complements other system analysis and design techniques by emphasizing participative development among system owners, users, designers, and builders. During JAD sessions for system design, the system designer will take on the role of facilitator for possibly several full-day workshops intended to address different design issues and deliverables.

Object-Oriented Development:

Object-oriented development focuses on the design of software components that mimic the real world. A component that adequately mimics the real world is much more likely to be used and reused. The approach emphasizes how a system operates, as opposed to analysis, which is concerned with what a system is capable of doing. One of the most important advantages in using an object-

Object-Oriented Development, continued:

oriented approach is the ability to reuse components. Traditional practices surrounding software development often mitigate against reuse. Short term goals are stressed because today's milestones must be achieved before any thought can be given to milestones that may be months or years away.

Borrowed or reused code is often code that has already been tested, and in the end, may translate into cost savings. Object-oriented development may make code reuse much easier but, the amount of actual reuse may still depend on the motivation of the project managers, designers and programmers involved. Code reuse can also lead to faster software development. Object-oriented software is easier to maintain because its structure is inherently decoupled. This usually leads to fewer side effects when changes have to be made. In addition, object-oriented systems may be easier to adapt and scale (i.e., large systems can be created by assembling reusable subsystems).

Typically, the object-oriented process follows an evolutionary spiral that starts with customer communication, where the problem is defined. The technical work associated with the process follows the iterative path of analysis, design, programming, and testing. The fundamental core concepts in object-oriented design involve the elements of *classes, objects, and attributes.* Understanding the definition and relationships of these elements is crucial in the application of object-oriented technologies.

It is recommend that the following object-oriented issues be well understood in order to form a knowledge base for the analysis, design, testing, and implementation of software using object-oriented techniques.

- What are the basic concepts and principles that are applicable to object-oriented thinking?
- How should object-oriented software projects be planned and managed?
- What is object-oriented analysis and how do its various models enable a software engineer to understand classes, their relationships and behavior?
- What is a 'use case' and how can it be applied to analyze the requirements of a system?
- How do conventional and object-oriented approaches differ?
- What are the components of an object-oriented design model?

Object-Oriented Development, continued:

- How are ‘patterns’ used in the creation of an object-oriented design?
- What are the basic concepts and principles that are applicable for testing of object-oriented software?
- How do testing strategies and test case design methods change when object-oriented software is considered?
- What technical metrics are available for assessing the quality of object-oriented software?

Work Product: The work products described in the Departmental software engineering methodology will be the same for many of the development techniques and it is the responsibility of the project manager to adapt the work products accordingly and document adaptations in the Project Plan.

Reference: Section 3.8, *Prepare Project Plan*, provides guidance for preparing a project plan.

This page intentionally left blank.